Optim Eng (2006) 7: 5–32
DOI 10.1007/s11081-006-6588-z

QPSchur: A dual, active-set, Schur-complement method for large-scale and structured convex quadratic programming

Roscoe A. Bartlett · Lorenz T. Biegler

Received: 20 March, 2002 / Revised: 23 March, 2005

Abstract We describe an active-set, *dual-feasible* Schur-complement method for quadratic programming (QP) with positive definite Hessians. The formulation of the QP being solved is general and flexible, and is appropriate for many different application areas. Moreover, the specialized structure of the QP is abstracted away behind a fixed KKT matrix called K_o and other problem matrices, which naturally leads to an object-oriented software implementation. Updates to the working set of active inequality constraints are facilitated using a dense Schur complement, which we expect to remain small. Here, the dual Schur complement method requires the projected Hessian to be positive definite for every working set considered by the algorithm. Therefore, this method is not appropriate for all QPs. While the Schur complement approach to linear algebra is very flexible with respect to allowing exploitation of problem structure, it is not as numerically stable as approaches using a QR factorization. However, we show that the use of fixed-precision iterative refinement helps to dramatically improve the numerical stability of this Schur complement algorithm. The use of the object-oriented QP solver implementation is demonstrated on two different application areas with specializations in each area; large-scale model predictive control (MPC) and reduced-space successive quadratic programming (with several different representations for the reduced Hessian). These results demonstrate that the QP solver can exploit application-specific structure in a computationally efficient and fairly robust manner as compared to other QP solver implementations.

Keywords Quadratic programming · Schur complement · Active-set · Dual space · Object oriented programming

R. A. Bartlett
Sandia National Laboratories, Albuquerque, NM 87185, USA
e-mail: rabartl@sandia.gov

L. T. Biegler (✉)
Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA
e-mail: lb01@andrew.cmu.edu

1. Introduction

Quadratic Programming (QP) methods are widely used in applications of control, finance and as a key subproblem in many nonlinear programming solvers. As larger and more challenging applications are considered, more attention is required for the development of efficient QP algorithms. In this study we consider the development of a flexible and efficient object-oriented QP method for large-scale use, and we compare this approach to some state-of-the-art QP solvers. In particular, our goal is to exploit large-scale problem structure and to handle the selection of working sets of active constraints efficiently.

Active set solvers for quadratic programming (see Fletcher, 1981, Chapter 10) can be classified into primal feasible (Gill et al., 1995; Gill, 1990; Betts and Frank, 1994) and dual feasible solvers (Goldfarb and Idnani, 1983; Schmid and Biegler, 1994; Powell, 1983). In the primal approach, a Phase I calculation is normally executed to find an initial feasible point, which is followed by a Phase II calculation where the KKT matrix is updated as constraints are added or dropped, while the algorithm reduces the objective function and maintains primal feasibility. The primal algorithm terminates successfully at a dual feasible point and strict convexity of the QP problem is not required for this approach. Dual feasible solvers, on the other hand, require a positive-definite projected Hessian but start with a dual feasible point that is usually computed cheaply (e.g. using the unconstrained minimum that initially ignores inequality constraints). In the dual approach, a KKT matrix is updated as constraints are added or dropped while the algorithm increases the objective function and maintains dual feasibility. The dual algorithm terminates successfully at a primal feasible point (i.e. because the duality gap is zero between the primal and dual formulations for convex QPs (Nash and Sofer, 1996)). We also mention that an alternative to active-set solvers is an interior point (or barrier) approach, such as the LOQO (Vanderbei, 1994) and OOQP[1] solvers.

Three popular approaches to factorize and update the KKT matrix are range space methods such as QPKWIK and ZQPCVX (Goldfarb and Idnani, 1983; Schmid and Biegler, 1994; Powell, 1983), null space methods such as QPOPT, Gill et al. (1995) and Schur complement methods such as SOCS (Gill, 1990; Betts and Frank, 1994). While the first two approaches often rely on variants of dense QR factorizations (and therefore possess good numerical stability properties), the Schur complement method exploits and performs only linear solves with a fixed initial KKT matrix (K_o) and seems to be ideally suited to exploit large, sparse and structured systems. The particular Schur complement implementation in Betts and Frank (1994) includes general sparse linear algebra solvers and has addressed very large QP subproblems successfully.

In this study, we also develop a Schur complement QP solver which enables us to effectively tailor linear algebra in an application-specific way for various large-scale, structured QPs. We assume positive-definite Hessians in the QP and therefore we opt for a dual feasible method, which we call QPSchur. The dual approach also allows great flexibility for the implementation of the QP constraints so as to exploit problem structure in an application-specific way. While current dual QP methods that use QR factorizations have good numerical stability, these dual QR methods can be inflexible with respect to being able to exploit problem structure. The fact that we adopt a Schur complement approach for linear algebra in order to be able to exploit application-specific structure comes at the expense of sacrificing numerical stability to some extent as compared to dense QR methods. However, we show that the use of fixed-precision iterative refinement can improve the numerical stability of Schur complement methods. Because QPSchur uses a dual approach, it differs from the

[1] http://www.cs.wisc.edu/∼swright/ooqp

primal Schur complement approaches of Gill (1990) and Betts and Frank (1994) and, to our knowledge, this is the first study that applies Schur complement approaches with a dual feasible algorithm. Therefore, a primary contribution of this work is the adaptation of the Schur complement approach for use in a dual QP algorithm. These adaptations are not (at least to us) immediately obvious and came about only after much thought, driven by numerical experience.

A second contribution of this work is a description of the QPSchur algorithm that lends itself naturally to an object-oriented (OO) implementation in C++. We also call this OO C++ implementation QPSchur. The QPSchur algorithm described here, however, is abstract and is more general than our current C++ QPSchur implementation. Through this approach, adding and dropping inequality constraints from the working set is done in a flexible manner. The QPSchur algorithm allows the structure of the underlying application area to be better exploited than with other dual QP methods, such as those based on the Goldfard and Idnani approach (Goldfarb and Idnani, 1983). The QPSchur algorithm allows great freedom in the definition and implementation for the Hessian matrix (by specializing the implementation of K_o in the Schur complement approach to linear algebra) and for the normals of the inequality constraints (as allowed by the dual algorithm). These properties make the resulting QP algorithm an ideal candidate for OO programming concepts which allow the exploitation of the structure for a number of applications. To demonstrate this ability to exploit application-specific problem structure, we consider several QP examples drawn from a broad range of applications.

The rest of this paper is organized as follows. The next section states the quadratic programming problem and presents the concept of the working set. Section 3 outlines the dual-feasible active-set algorithm consistent with Goldfarb and Idnani (1983). Section 4 presents a review of the Schur complement QP method consistent with Gill (1990) and describes the basic factorization updates and step computations for adding and dropping active constraints. Section 5 describes the combined Schur complement dual active-set QP algorithm QPSchur which is the new contribution of this work. This section includes detailed calculations for the QP steps, treatment of degeneracy, safeguards in the presence of ill-conditioning and roundoff error and a warm start algorithm based on an initial guess for the working set. Section 6 demonstrates the benefits of the QP algorithm and its object-oriented implementation on applications drawn from process control and successive quadratic programming. Finally, Section 7 concludes the paper and presents areas for future work.

2. The quadratic programming problem and the working set

The general formulation for the QP to be solved is

$$\min_{x \in \mathbb{R}^n} \quad g^T x + \tfrac{1}{2} x^T G x \tag{1}$$

$$\text{s.t.} \quad \begin{bmatrix} x_L \\ c_L \end{bmatrix} \leq \begin{bmatrix} I \\ A_c^T \end{bmatrix} x \leq \begin{bmatrix} x_U \\ c_U \end{bmatrix} \tag{2}$$

where: $x, x_L, x_U \in \mathbb{R}^n$, $g \in \mathbb{R}^n$, $G = G^T \in \mathbb{R}^{n \times n}$ is positive definite, $A_c \in \mathbb{R}^{n \times m}$ and $c_L, c_U \in \mathbb{R}^m$. The QPSchur algorithm attempts to solve (1)–(2) for the first-order KKT optimality conditions, which include feasibility of the constraints (2) and linear dependence of the gradients

$$g + Gx + A_c \lambda + \mu = 0. \tag{3}$$

The solution consists of the unknown primal variables x and the multipliers λ for $c_L \leq A_c^T x \leq c_U$ and μ for $x_L \leq x \leq x_U$. When appropriate, to allow a more abstract handling of the inequality constraints in (2), we define the aggregate matrix $A = [I \quad A_c]$ and its aggregate multipliers $\nu^T = [\mu^T \quad \lambda^T]$. We note that at the optimum that the j^{th} element of ν will satisfy the complementarity conditions: $\nu_j \geq 0$ for an active upper bound, $\nu_j \leq 0$ for an active lower bound, and $\nu_j = 0$ for an inactive constraint. Active-set QP algorithms, such as QPSchur, search through different working sets of active constraints until the above optimality conditions are met or the QP is found to be infeasible. In active-set QP algorithms, equality-constrained QPs of the form

$$\min_{x \in \mathbb{R}^n} \quad g^T x + \tfrac{1}{2} x^T G x \tag{4}$$

$$\text{s.t.} \quad A_w^T x = b_w \tag{5}$$

are solved for each instance of the working set where A_w corresponds to the constraint normals for the working subset of (2) and b_w corresponds to either the upper or lower bounds in (2) for the working set.

The dual active-set QP algorithm is described in the next section. This dual QP algorithm, as well as every other active-set QP algorithm, requires the solution of linear systems related to the KKT system for equality-constrained QPs of the form (4)–(5).

3. Overview of the dual active-set QP algorithm for adjusting the working set

The dual active-set QP algorithm maintains iterates that are dual feasible (i.e. the multipliers ν have the correct sign as described above) but primal infeasible (i.e. one or more of the inequality constraints in (2) are violated by more than a prescribed tolerance) and one form of this method, as described in Goldfarb and Idnani (1983), is outlined here. At each major iteration in the dual algorithm, a violated inequality constraint from (2) with index $j^{(+)}$ and the violated bound $b_{j^{(+)}}$ is selected to add to the current working set of active inequality constraints. From now on we will refer to this violated constraint simply by its index $j^{(+)}$ (i.e. violated constraint $j^{(+)}$). The next phase in a dual-algorithm iteration considers the addition of the violated constraint $j^{(+)}$ to the working set by computing steps s^x and s^ν for the current estimates for the primal x and dual ν variables, respectively, that define the updates

$$(x)^+ = x + \beta t s^x, \tag{6}$$

$$(\nu)^+ = \nu + \beta t s^\nu. \tag{7}$$

The value of the step $t = t^P$ in (6)–(7), known as the primal step length, corresponds to the the solution of the KKT system with the addition of the constraint $j^{(+)}$ to the working set. The sign

$$\beta \equiv \begin{cases} +1 & \text{if } (A_{(:,j^+)})^T x > b_{j^{(+)}} : \text{violated upper bound} \\ -1 & \text{if } (A_{(:,j^+)})^T x < b_{j^{(+)}} : \text{violated lower bound} \end{cases} \tag{8}$$

in (6)–(7) can be inferred a priori since the objective function must increase when adding the violated constraint $j^{(+)}$. The multiplier value $\gamma^+ = \beta t^P$ for the new constraint $j^{(+)}$ is

computed as well. Before the new constraint $j^{(+)}$ can be added to the active-set, the dual step length $t = t^D$ must be computed. The dual step length t^D is the maximum value of $t = t^D$ such than $(v)^+$ remains dual feasible for the update formula (7). If $t^D < t^P$, then the multiplier that becomes $v_{j^{(-)}} = 0$ determines the constraint $j^{(-)}$ that will be removed from the working set. If the new constraint $j^{(+)}$ is linearly dependent with those in the current working set, then the solution to the KKT system for the augmented working set is undefined. However, in this case, the steps s^x and s^v are still defined and the step s^v is used to select another constraint, with index $j^{(-)}$, from the current working set to drop. In essence this replaces the dropped constraint $j^{(-)}$ with the new constraint $j^{(+)}$.

This basic logic of the dual algorithm is described below in Algorithm 3.1.

Algorithm 3.1. *Dual Active-Set QP Algorithm*

- **Initialization** *Find an initial working set of active constraints that gives a nonsingular KKT system and a dual feasible point (i.e. with (3) satisfied and the correct signs for the multipliers).*
- **Dual Iterations**

for iter $= 1 \ldots$ `max_iter`

1. **PICK_VIOLATED_CONSTRAINT**
   - (a) *Pick an inequality constraint from (2) with index $j^{(+)}$ that is violated by more than some tolerance.*
   - (b) *Check for convergence. If no violated constraints are found, STOP, the current point is optimal.*
2. **COMPUTE_SEARCH_DIRECTION**
   - (a) *Compute search directions for the primal and dual variables s^x and s^v, respectively.*
3. **COMPUTE_STEP_LENGTHS**
   - (a) *Compute the primal step length t^P (note, $t^P = \infty$ if constraint $j^{(+)}$ is linearly dependent with the current working set).*
   - (b) *Compute the dual step length t^D (i.e. maximum step for dual feasibility) and the limiting multiplier index $j^{(-)}$ for the update rule (7) that keeps the updated multipliers v^+ for the current working set from changing sign.*
4. **TAKE_STEP**
   - (a) *If $t^P = \infty$ and $t^D = \infty$, then label the QP as infeasible, as the added constraint $j^{(+)}$ is inconsistent. Terminate algorithm!*
   - (b) *Elseif $t^D < t^P$, take a Dual step (i.e. $t^P = \infty$) or Partial Primal-Dual step (i.e. t^P is finite). For the dual step, $j^{(+)}$ is linearly dependent and we require a drop/add operation where the constraint $j^{(-)}$ is replaced with the incoming constraint $j^{(+)}$. For the partial primal-dual step, $j^{(+)}$ is linearly independent and we drop the constraint $j^{(-)}$. Update factorizations for this add/drop.*
     Goto Step {2}
   - (c) *Elseif $t^P \leq t^D$, take a Full Primal-Dual step using (6)–(7) and add constraint $j^{(+)}$ to the working set. Update factorizations for adding $j^{(+)}$ to the working set if not done already.*

endfor

Key theory for the behavior and convergence of the algorithm can be found in Goldfarb and Idnani (1983) and Bartlett (2001). In particular,

- For every iteration where a linearly independent violated constraint is selected to add to the working set, a nonzero step for the primal variables $s^x \neq 0$ will be computed and will result in an increase in the strictly convex objective function.
- For iterations where a linearly dependent constraint is added to the working set, the step for the primal variables s^x will be zero but the step for the dual variables s^v will not be zero.
- If the constraints are consistent, then when a linearly dependent constraint is added, a constraint will be selected to be dropped from the working set. This means that $t^D < \infty$ will be computed in Step $\{3\}$ and therefore Step $\{4\}$ will never be executed unless the constraints are infeasible. In addition, after this dropped constraint $j^{(-)}$ is replaced by the newly added constraint $j^{(+)}$, the new KKT system after the drop/add will be nonsingular and the dropped constraint $j^{(-)}$ will become strictly feasible.

It is easy to show that, in exact arithmetic, the strictly convex objective function always increases for every major iteration (see Bartlett (2001)) and therefore the same working set will never be visited twice. Since there are only a finite number of permutations of active constraints in the working set possible, the algorithm must terminate in a finite number of iterations. With ill-conditioning and roundoff errors, however, a floating-point implementation of the algorithm may cycle. Safeguards against this are discussed in the context of the QPSchur algorithm in Section 5.3.

The linear algebra used to compute the steps s^x and s^v can be implemented in a variety of ways. QR factorizations described in Goldfarb and Idnani (1983) possess superior numerical stability properties but are often inflexible with regard to exploiting application-specific structure. Therefore we consider a Schur complement approach for handling the linear algebra in a flexible way. The basics of the Schur complement approach for the linear algebra are outlined in the next section and then the combined Schur complement dual QP algorithm QPSchur is presented in Section 5.

4. Overview of linear algebra for the Schur complement active-set QP method

As stated earlier, one particularly flexible approach for performing the linear algebra needed to solve (4)–(5) associated with the current working set is the Schur complement method. Here we provide an overview of the Schur complement method consistent with Gill (1990). The purpose of the following overview is to establish nomenclature and to set the context for describing the computations in the QPSchur algorithm discussed in Section 5.

To lay the groundwork for the Schur complement QP method, we first consider an initial KKT system involving only active variable bounds (i.e. fixed[2] variables) which defines a matrix K_o. We then describe in Section 4.2 how the Schur complement method uses this initial KKT system and K_o to solve KKT systems for changes to the working set using a Schur complement matrix S. We then present in Section 4.3 how the Schur complement

[2] Here we use the term "fixed" to describe variables for which either their upper or lower bounds are part of the current working set. This is consistent with the terminology used in Gill (1990).

S is updated and downdated for individual changes to the working set. Finally, details on implementation options for the factorization Schur complement S and on updating the factors of S for single changes to the Schur complement are described in Section 4.4.

4.1. Initial KKT system

The Schur complement method starts with a working set of initially free (x^R) and fixed (x^X) variables. Here, x is partitioned using the permutation matrix $Q = [Q^R \quad Q^X]$, with $Q^R \in \mathbb{R}^{n \times n^R}$ and $Q^X \in \mathbb{R}^{n \times n^X}$, as

$$Q^T x = \begin{bmatrix} (Q^R)^T \\ (Q^X)^T \end{bmatrix} x = \begin{bmatrix} (Q^R)^T x \\ (Q^X)^T x \end{bmatrix} = \begin{bmatrix} x^R \\ x^X \end{bmatrix} \tag{9}$$

and $b^X \in \mathbb{R}^{n^X}$ defines the initial active variable bounds, where $b_l^X = [(Q^X)^T x_L]_l$ or $[(Q^X)^T x_U]_l$, for $l = 1 \ldots n^X$. Given the permutations in (9), initial KKT system for the QP is given by

$$\begin{bmatrix} G^{RR} & G^{RX} & \\ (G^{RX})^T & G^{XX} & I \\ & I & \end{bmatrix} \begin{bmatrix} x^R \\ x^X \\ \mu^X \end{bmatrix} = \begin{bmatrix} -g^R \\ -g^X \\ b^X \end{bmatrix} \tag{10}$$

where $G^{RR} = (Q^R)^T G(Q^R)$, $G^{RX} = (Q^R)^T G(Q^X)$, $G^{XX} = (Q^X)^T G(Q^X)$, $g^R = (Q^R)^T g$, $g^X = (Q^X)^T g$ and $\mu^X = (Q^X)^T \mu$. The dual QP method requires that this initial working set result in multipilers μ^X that are dual feasible.

This system can be solved in three blocks by first setting $x^X = b^X$, then solving the linear system

$$G^{RR} x^R = -g^R - G^{RX} b^X \tag{11}$$

followed by setting $\mu^X = -g^X - (G^{RX})^T x^R - (G^{XX}) b^X$. We denote the initial KKT system in (11) as $K_o y_o = f_o$ (where $K_o = G^{RR}$, $y_o = x^R$ and $f_o = -g^R - G^{RX} b^X$) which defines the initial solution vector

$$y_o = K_o^{-1} f_o. \tag{12}$$

The efficiency of the Schur complement method depends on the utilization of the initial solution y_o computed in (12) which requires that the definition of f_o not change during the course of the algorithm.

For the dual QP method to succeed, G must be symmetric positive definite (s.p.d.) when projected into *any* of the working sets examined by a QP algorithm. As result, we require that G be s.p.d. (which occurs in our applications).

With the introduction of the initial KKT system, we now consider the details of modifying the working set with the Schur complement method. In the next section, the computation of the Schur complement is presented in response to a set of changes (for adding or dropping constraints) to the working set. We then present the updating strategy for changes that involve the addition and subtraction of single constraints in Section 4.3.

4.2. Representing the KKT system with changes to the initial working-set

For a set of changes from the initial working set (9), the augmented KKT system takes the form

$$\begin{bmatrix} K_o & U \\ U^T & V \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} f_o \\ d \end{bmatrix}. \tag{13}$$

Here the matrices U, V and vectors z, d arise from changes in the working set due to addition of constraints or freeing initially fixed variables. The structure of these matrices and vectors is detailed below. System (13) can be solved by applying block Gaussian elimination by pivoting on the nonsingular block K_o (which is guaranteed to be s.p.d. if G is s.p.d.). This leads to the formation of the Schur complement matrix

$$S \equiv V - U^T K_o^{-1} U. \tag{14}$$

Given the above Schur complement S, the system (13) can then be solved as

$$z = S^{-1}(d - U^T y_o) \tag{15}$$

followed by

$$y = K_o^{-1}(f_o - Uz), \tag{16}$$

where y_o is defined in (12). The Schur complement matrix S in (14) is guaranteed to be nonsingular if K_o and the full KKT matrix in (13) are nonsingular (Carlson, 1986).

The primary advantage of the Schur complement approach is that it allows great flexibility in the representation and implementation of the initial KKT matrix K_o since the method only performs linear solves with K_o. Since only linear solves with K_o are required the implementation of this matrix operator is completely arbitrary and can be adapted to exploit the specific structure and properties of a particular class of QPs.

We now consider several different types of changes to the initial working set that represent (13). First we define the integer q^F as the number of variables initially fixed and currently not at their initial bound, the integer q^D as the number of variables initially fixed and still remain fixed at their original bounds, and the integer $q^{(+)}$ as the number of initially free constraints from (2) added to the working set. The last type of change to the working set is specialized to double-sided variable bounds. It is defined by the integer q^C as the number of variables initially fixed, then freed, then fixed to their opposite bounds. Note that in order for a variable that is initially fixed to one of its bounds to be later fixed to its other bound, the variable must first be freed from its initial bound. Therefore, the number of changes q^C satisfies $q^C \leq q^F$. This set of changes to the working set is discussed in more detail below. For the rest of this discussion we will use the integers q^F, q^D, $q^{(+)}$ and q^C to indicate these four different types of changes to the working set.

The dimension of the Schur complement S in (14) will be shown to be $S \in \mathbb{R}^{q \times q}$ where $q = q^F + q^{(+)} + q^C$. We now define a set of mapping matrices that can be used to define sets of variables and active constraints in the working set. First, define the mapping matrices $Q^F \in \mathbb{R}^{n^X \times q^F}$, $Q^D \in \mathbb{R}^{n^X \times q^D}$ and $Q^C \in \mathbb{R}^{q^F \times q^C}$ that partition the initially fixed variables x^X into subsets where $x^F = (Q^F)^T x^X$ are initially fixed variables that are currently not fixed at

their initial bound, $x^D = (Q^D)^T x^X$ are initially fixed variables that are currently still fixed at their initial bounds, and $x^C = (Q^C)^T x^F$ are initially fixed variables that where freed and are currently fixed at the opposite bound. From the above definitions, the current status in the working set for initially fixed variables must satisfy $n^X = q^F + q^D$ and $q^C \leq q^F \leq n^X$.

The last permutation matrix that we define is $Q^{(+)} \in \mathbb{R}^{m \times q^{(+)}}$ which specifies $A^{(+)} = AQ^{(+)}$ which are the constraint normals in (2) in the current working set that are not associated with initially fixed variables. Note that since the total number of constraints in the working set must not exceed the total number of variables, then it is required that $n^X - q^F + q^C + q^{(+)} \leq n$.

At this point we also define other quantities that are needed to specify the augmented KKT system for the current working set. We define the vector b to signify the relevant upper or lower bound in (2), such that $b^{(+)} = (Q^{(+)})^T b$ selects the active bounds and b^X_* is used to represent the opposite set of initial active bounds b^X. For example, if $b^X_l = (x^X_L)_l$ then $(b^X_*)_l = (x^X_U)_l$ (and vice versa). Given b^X and b^X_*, we have $b^C_* \equiv (Q^C)^T(Q^F)^T b^X_*$, $b^C_* \equiv (Q^C)^T(Q^C)^T b^X_*$ and $b^D \equiv (Q^D)^T b^X$. We also define $\nu^{(+)}$ to be the subset of ν that contains multipliers for the constraints from $(x_L)^R \leq x^R \leq (x_U)^R$ and $c_L \leq A_c^T x \leq c_U$ in the working set. The multipliers for initially fixed variables that are currently still in the working set are contained in $\mu^C = (Q^C)^T(Q^F)^T \mu^X$ and $\mu^D = (Q^D)^T \mu^X$.

Now that a set of changes from the initial working set has been defined, the derivation of (13) is started by writing the KKT system for the current working set as

$$\begin{bmatrix} G^{RR} & G^{RF} & G^{RD} & A^{(+)R} & & \\ (G^{RF})^T & G^{FF} & G^{FD} & A^{(+)F} & Q^C & \\ (G^{RD})^T & (G^{FD})^T & G^{DD} & A^{(+)D} & & I \\ (A^{(+)R})^T & (A^{(+)F})^T & (A^{(+)D})^T & & & \\ & (Q^C)^T & & & & \\ & & I & & & \end{bmatrix} \begin{bmatrix} x^R \\ x^F \\ x^D \\ \nu^{(+)} \\ \mu^C \\ \mu^D \end{bmatrix} = \begin{bmatrix} -g^R \\ -g^F \\ -g^D \\ b^{(+)} \\ b^C_* \\ b^D \end{bmatrix}. \tag{17}$$

At some point during the QP algorithm, some of the initially fixed variables x^X may need to be freed from their initial bounds. To do this, the algorithm computes the change p^X from the initial bound and sets $x^X = p^X + b^X$. Only elements p^X_l are computed for those x^X_l that do not remain at the initial bound. All other elements in p^X are implicitly zero. Permuting (17) and substituting $x^F \equiv p^F + b^F$ (where $p^F \equiv (Q^F)^T p^X$ and $b^F \equiv (Q^F)^T b^X$) leads to the system represented as

$$\begin{bmatrix} K_o & \tilde{U} \\ \tilde{U}^T & \tilde{V} \end{bmatrix} \begin{bmatrix} \tilde{y} \\ \tilde{z} \end{bmatrix} = \begin{bmatrix} f_o \\ \tilde{d} \end{bmatrix} \tag{18}$$

where

$$\tilde{U} = [G^{RF} \;\; A^{(+)R} \;\; 0], \tag{19}$$

$$\tilde{V} = \begin{bmatrix} G^{FF} & A^{(+)F} & Q^C \\ (A^{(+)F})^T & 0 & 0 \\ (Q^C)^T & 0 & 0 \end{bmatrix}, \tag{20}$$

$$\tilde{d} = \begin{bmatrix} -g^F - (Q^F)^T G^{XX} b^X \\ b^{(+)} - (A^{(+)X})^T b^X \\ b^C_* - b^C \end{bmatrix}, \tag{21}$$

$$\tilde{z} = \begin{bmatrix} p^F \\ \nu^{(+)} \\ \mu^C \end{bmatrix}, \tag{22}$$

and $G^{RX} b^X = G^{RF} b^F + G^{RD} b^D$, $(Q^F)^T G^{XX} b^X = G^{FF} b^F + G^{FD} b^D$ and $(A^{(+)X})^T b^X = (A^{(+)F})^T b^F + (A^{(+)D})^T b^D$. Following the solution of the permuted augmented KKT system in (18) the multipliers μ^D are computed from (17) as:

$$\mu^D = -g^D - (G^{RD})^T x^R - (G^{FD})^T (p^F + b^F) - G^{DD} b^D - A^{(+)D} \nu^{(+)}. \tag{23}$$

For the augmented KKT systems, we distinguish the arbitrary ordering of changes to the working set in (13) and the partitioned changes to the working set in (18) using the tilde '~'. These two KKT systems are equivalent, as one is a permutation of the other. To allow for the changes to the working set to occur in any arbitrary order as shown in (13), we define a permutation matrix $\tilde{P} \in \mathbb{R}^{q \times q}$ which gives $U = \tilde{U}\tilde{P}$, $V = \tilde{P}^T \tilde{V} \tilde{P}$, $d = \tilde{P}^T \tilde{d}$ and $z = \tilde{P}^T \tilde{z}$. The quantities U, V, d and z can be expressed in terms of the original problem matrices and vectors (see Bartlett (2001)). This permutation matrix is used to define individual changes to the working set in the next section.

4.3. Single changes to the working set and updating the Schur complement

We now describe the individual changes needed for the augmented KKT system and the Schur complement whenever a constraint is added or dropped from the working set. These changes to the working set require the dimension of the augmented KKT system and the Schur complement $S \to \bar{S}$ to increase ($\bar{q} = q + 1$) or to decrease ($\bar{q} = q - 1$). From now on the bar '¯' will be used to denote the quantities for a single change to the working set. In the following discussion we first describe changes to the working set that expand the Schur complement $S \to \bar{S}$ in Section 4.3.1 and then changes that contract $S \to \bar{S}$ in Section 4.3.2.

4.3.1. Changes that expand the Schur complement

Consider a change to the working set of active constraints where the dimension of the augmented KKT system (and therefore the Schur complement) increases (i.e. $\bar{q} = q + 1$). The new augmented KKT system is given as

$$\begin{bmatrix} K_o & \bar{U} \\ \bar{U}^T & \bar{V} \end{bmatrix} \begin{bmatrix} y \\ \bar{z} \end{bmatrix} = \begin{bmatrix} f_o \\ \bar{d} \end{bmatrix} \tag{24}$$

where $\bar{U} = [U \;\; u^p]$, $\bar{V} = \begin{bmatrix} V & v^p \\ (v^p)^T & \sigma \end{bmatrix}$, $\bar{d} = \begin{bmatrix} d \\ d^p \end{bmatrix}$, $\bar{z} = \begin{bmatrix} z \\ z^p \end{bmatrix}$.

The new augmented Schur complement is given as

$$\bar{S} = \bar{V} - \bar{U}^T K_o^{-1} \bar{U} = \begin{bmatrix} S & \bar{t} \\ \bar{t}^T & \bar{\alpha} \end{bmatrix} \tag{25}$$

where $\bar{t} = v^p - U^T r$, $\bar{\alpha} = \sigma - (u^p)^T r$, and $r = K_o^{-1} u^p$.

The corresponding values of the vectors u^p and v^p and the scalars σ, d^p and z^p in (24) and (25) are given below. Changes to the working set that expand the Schur complement include $\bar{q}^F = q^F + 1$ for an initially fixed variable $x_l^X = b_l^X$ now being freed, $\bar{q}^{(+)} = q^{(+)} + 1$ for adding a constraint $(A_{(:,j)})^T x = b_j$ to the working set, and $\bar{q}^C = q^C + 1$ for an initially fixed variable $x_l^X = b_l^X$ that was freed and is now being fixed to the opposite bound $x_l^X = (b_*^X)_l$.

Here we use the colon notation $A_{(i,:)}$ and $A_{(:,j)}$ to signify the ith row and the jth column, respectively, of a matrix A. The quantities u^p, v^p, σ, d^p and z^p can be derived by inspecting (19)–(22) for the unpermuted system (18) and then applying the permutation matrix $\tilde{P}$. In the partitioned system, $\tilde{u}^p$ can be determined by inspection of (19) for each of the changes to the working set

$$\tilde{u}^p = \begin{cases} G^{RX}_{(:,l)} & : \text{for } \bar{q}^F = q^F + 1 \\ A^R_{(:,j)} & : \text{for } \bar{q}^{(+)} = q^{(+)} + 1 \\ 0 & : \text{for } \bar{q}^C = q^C + 1 \end{cases} \tag{26}$$

and the permuted quantity is $u^p = \tilde{u}^p$.

By inspecting (20) it is clear that

$$\tilde{v}^p = \begin{cases} \begin{bmatrix} (Q^F)^T G^{XX}_{(:,l)} \\ (Q^{(+)})^T (A^X_{(:,l)})^T \\ 0 \end{bmatrix} & : \text{for } \bar{q}^F = q^F + 1 \\ \begin{bmatrix} (Q^F)^T A^X_{(:,j)} \\ 0 \\ 0 \end{bmatrix} & : \text{for } \bar{q}^{(+)} = q^{(+)} + 1 \\ \begin{bmatrix} e_k \\ 0 \\ 0 \end{bmatrix} & : \text{for } \bar{q}^C = q^C + 1 \end{cases} \tag{27}$$

$$\tilde{\sigma} = \begin{cases} G^{XX}_{(l,l)} & : \text{for } \bar{q}^F = q^F + 1 \\ 0 & : \text{for } \bar{q}^{(+)} = q^{(+)} + 1 \\ 0 & : \text{for } \bar{q}^C = q^C + 1 \end{cases} \tag{28}$$

The quantities v^p and σ are determined as

$$\bar{V} = \begin{bmatrix} \tilde{P}^T & \\ & 1 \end{bmatrix} \begin{bmatrix} \tilde{V} & \tilde{v}^p \\ (\tilde{v}^p)^T & \tilde{\sigma} \end{bmatrix} \begin{bmatrix} \tilde{P} & \\ & 1 \end{bmatrix} = \begin{bmatrix} \tilde{P}^T \tilde{V} \tilde{P} & \tilde{P}^T \tilde{v}^p \\ (\tilde{v}^p)^T \tilde{P} & \tilde{\sigma} \end{bmatrix} = \begin{bmatrix} V & v^p \\ (v^p)^T & \sigma \end{bmatrix} \tag{29}$$

From (27) note that $v^p = e_k \in \mathbb{R}^q$ for $\bar{q}^C = q^C + 1$, where k is the index of the row and column in S where the variable x_l^X was earlier freed from its initial bound, and e_k is the unit vector. For this change, our approach and the one in Gill (1990) preserves the initial right-hand side f_o and therefore the initial KKT solution $y_o = K_o^{-1} f_o$ in (12). Changing the definition of f_o to set an initially fixed variable to its other bound would require a solve with K_o in the recomputation of $y_o = K_o^{-1} f_o$. However, by augmenting the KKT system by using (20) instead, we avoid this additional solve with K_o in augmenting $S \to \bar{S}$, since $u^p = 0$ for this case (i.e. $r = K_o^{-1} u^p$ is skipped in (25)). Therefore, this approach of augmenting the KKT system for $\bar{q}^C = q^C + 1$ requires fewer solves with K_o, but at the cost of a larger Schur complement.

Finally, d^p and z^p can be determined from (21) and (22) as

$$d^p = \begin{cases} -g_l^X - \left(G^{XX}_{(l,:)}\right)^T b^X & : \text{for } \bar{q}^F = q^F + 1 \\ b_j - \left(A^X_{(:,j)}\right)^T b^X & : \text{for } \bar{q}^{(+)} = q^{(+)} + 1 \\ (b_*^X)_l - b_l^X & : \text{for } \bar{q}^C = q^C + 1 \end{cases} \tag{30}$$

$$z^p = \begin{cases} p_l^X & : \text{for } \bar{q}^F = q^F + 1 \\ \nu_j & : \text{for } \bar{q}^{(+)} = q^{(+)} + 1 \\ \mu_l^X & : \text{for } \bar{q}^C = q^C + 1 \end{cases} \tag{31}$$

As shown in (25), the major computational work in computing new terms in the augmented Schur complement involves one solve with K_o and a matrix-vector product $U^T r$. Also, the work required to update or recompute the factorization of $S \to \bar{S}$ may be significant if q is large (especially if S is indefinite and therefore, in general, requires a complete $O(q^3)$ refactorization).

4.3.2. Changes that shrink the Schur complement

Changes to the working set that shrink the Schur complement include $\bar{q}^F = q^F - 1$ (for an initially fixed variable that was freed and is now being fixed back to is original bound), $\bar{q}^C = q^C - 1$ (for an initially fixed variable that was fixed to its opposite bound and is now being freed), and $\bar{q}^{(+)} = q^{(+)} - 1$ (for removing an extra constraint that was earlier added to working set). These changes to the working set involve undoing a previous change where the KKT system was augmented and therefore the dimension of the Schur complement is decreased $\bar{q} = q - 1$ to remove this earlier change. This is accomplished by removing the row and column k corresponding to the previous change which is being erased. This change $S \to \bar{S}$ is illustrated as

$$\bar{S} = \begin{bmatrix} S_{(1:k-1,1:k-1)} & S_{(1:k-1,k+1:q)} \\ S_{(k+1:q,1:k-1)} & S_{(k+1:q,k+1:q)} \end{bmatrix}. \tag{32}$$

The only computational work involved here is the work required to update the factorization of S which is discussed next.

4.4. Factoring the Schur complement and updating its factorization

It can be shown that if the projected Hessian for the KKT system for the current working set in (13) is s.p.d. then the inertia of the Schur complement S is $\text{In}(S) = (q^{(+)} + q^C, 0, q^F)$, which gives the number of negative, zero and positive eigenvalues in S, respectively (Gill, 1990). The implications of this are that if no initially fixed variables are freed (i.e. $q^F = 0$), then $-S$ is s.p.d.; otherwise S may be indefinite.

Our implementation of S uses either a s.p.d. Cholesky factorization for $-S$ or a symmetric indefinite Bunch-Kaufman factorization (Golub, 1996) for S. When we know that $-S$ should be s.p.d. (i.e. when $q^F = 0$) then we use the Cholesky factorization. The Cholesky factorization is easily and stably updated when symmetric rows and columns are added or removed at a cost of $O(q^2)$ flops per update. On the other hand, when we know that S should be indefinite (i.e. when $q^{(+)} + q^C > 0$ and $q^F > 0$) we use a dense Bunch-Kaufman factorization. In the current implementation, we recompute the Bunch-Kaufman factorization from scratch for every change in S, which results in $O(q^3)$ flops. For larger q, this means that the updating an indefinite S is $O(q^3)$ and is much more expensive than for a negative definite S which is $O(q^2)$. In Section 7 we discuss as future work the potential for reducing the cost for an indefinite S by updating the factors of the symmetric indefinite factorization.

5. Schur complement dual active-set QP algorithm QPSchur

In this section we describe an integration of the dual algorithm for adjusting the working set outlined in Section 3 and the Schur complement method for handling linear algebra outlined in Section 4 to produce a flexible and effective algorithm for solving strictly convex QPs which we call QPSchur. All previously known uses of the Schur complement method for linear algebra have been used in a primal active-set QP algorithm (Gill, 1990; Betts and Frank, 1994). The behavior of a dual active-set QP algorithm, such as used for QPSchur, is significantly different from a primal QP algorithm and these differences impact the linear algebra computations. The primary linear algebra computations that we describe here are the computation of the steps for the primal and dual variables s^x and s^v shown in (6) and (7), respectively, and the update of the Schur complement. The computation of the steps and the close relationship to updating the Schur complement is described in the next section. This is followed in Section 5.2 with a description of a warm-start algorithm where an initial guess of the optimal working set is used to find an initial dual-feasible point. Lastly, Section 5.3 briefly discusses numerical accuracy issues and iterative refinement.

5.1. Computation of steps for primal and dual variables

The QPSchur algorithm does not directly deal with vectors for the variables x and $v^T = [\mu^T \quad \lambda^T]$ but instead components for these vectors are buried in several different vectors computed in the Schur complement method. For example, the current estimate of the primal solution x can be constructed from various components from z (15), y (16) and in b^X (9). The current estimate of the multipliers μ is buried in z and μ^D (23). Finally, the current estimate of the multipliers λ is buried in z. Reconstructing x, μ and λ from z, y, μ^D and b^X

is primarily a book-keeping operation. Closed-form expressions for x and $\nu^{(+)}$ in terms of the various permutation and mapping matrices defined above are given in Bartlett (2001). From here on we will deal with quantities from the perspective of the Schur complement unknowns but the connection to the original primal and dual variables x and ν should be understood.

To compute the steps for the primal and dual variables s^x and s^ν shown in Algorithm 3.1 Step {2}, we note that the Schur complement method computes the native vectors z, y and μ^D which determine x and ν. As shown in Section 5.1.1, steps s^y and s^z are computed by the Schur complement method that define the update rules

$$y^+ = y + \beta t s^y, \tag{33}$$

$$z^+ = z + \beta t s^z, \tag{34}$$

where t is the step determined by Algorithm 3.1 and β is defined in (8). Once the algorithm has computed s^z and s^y, then $s^R = s^y$ from (11)–(12) and $s^{\nu^{(+)}}$, s^C, and s^F are obtained by picking out the appropriate entries from s^z. Also, from (23), the expression for $(\mu^D)^+$ is given by

$$(\mu^D)^+ = -g^D - (G^{RD})^T (x^R)^+ - (G^{FD})^T ((p^F)^+ + b^F) - G^{DD} b^D - A^{(+)D} \nu^{(+)} \tag{35}$$

To compute s^D, we substitute $(x^R)^+ = x^R + \beta t s^R$, $(p^F)^+ = p^F + \beta t s^F$ and $(\nu^{(+)})^+ = \nu^{(+)} + \beta t s^{\nu^{(+)}} + \beta t e_{j^{(+)}}$ into (35) and obtain $s^D = -(G^{RD})^T s^R - (G^{FD})^T s^F - A^{(+)D}(s^{\nu^{(+)}} + e_{j^{(+)}})$.

5.1.1. Computation of s^y and s^z

There are two primary approaches for computing steps s^y and s^z and determining the method that results in the least numerical computation depends on the nature of the incoming constraint $j^{(+)}$ and the current state of the working set. In all but one case it will become obvious which approach requires the least computational expense.

The first approach for computing the steps s^y and s^z is to update the Schur complement by (24) and (25), followed by the computation of the solutions $\bar{z}^+$ and y^+ using

$$\bar{z}^+ = \bar{S}^{-1}(\bar{d} - \bar{U}^T y_o) \tag{36}$$

$$y^+ = K_o^{-1}(f_o - \bar{U}(\bar{z})^+) \tag{37}$$

and then setting

$$s^z = ((\bar{z})^+_{(1:q)} - z)/(\gamma^+ \beta) \tag{38}$$

$$s^y = ((y)^+ - y)/(\gamma^+ \beta). \tag{39}$$

where γ^+ is either the last component of $\bar{z}$ or $\bar{\mu}^D$. Note that this first approach of attempting to first update the Schur complement for the addition of the constraint $j^{(+)}$ may not succeed since the augmented KKT system may be singular if the constraint $j^{(+)}$ is linearly dependent with the current working set. Therefore, a second approach to compute s^y and s^z is required when the augmented KKT system is singular.

This second approach uses the nonsingular Schur complement S for the current working set. In this approach we start by defining the augmented KKT system

$$\begin{bmatrix} K_o & U & u^a \\ U^T & V & v^a \\ (u^a)^T & (v^a)^T & \end{bmatrix} \begin{bmatrix} y^+ \\ z^+ \\ \gamma^+ \end{bmatrix} = \begin{bmatrix} f_o \\ d \\ d^a \end{bmatrix} \tag{40}$$

where using (26), (29) and (30) we obtain

- $u^a = u^p$, $v^a = v^p$, and $d^a = d^p$, for $(\bar{q}^{(+)} = q^{(+)} + 1)$,
- $u^a = 0$, $v^a = v^p$, and $d^a = d^p$, for $(\bar{q}^C = q^C + 1)$, and
- $u^a = 0$, $v^a = e_{k^{(+)}}$, and $d^a = 0$, for $(\bar{q}^F = q^F - 1)$.

Note that for the case $\bar{q}^F = q^F - 1$, where an initially fixed variable that was freed is now being fixed back to its original bound, the above KKT system is augmented in (40) instead of shrunk when updating the Schur complement in (32). As described in Section 4.3.2, shrinking the Schur complement first may make it singular if S is indefinite. For this case, the constraint $e_k^T z = 0$ can be added to the working set where $k \in [1, q]$ is the row and column in S where the variable x_l^X was earlier freed from its initial bound. This constraint gives $e_k^T z = p_l^X = 0$ and therefore the algorithm fixes $x_l^X = p_l^X + b_l^X = b_l^X$.

Next, we substitute (33), (34) and $\gamma^+ = 0 + \beta t$ into (40) and add the shown zero vector to obtain

$$\begin{bmatrix} K_o & U & u^a \\ U^T & V & v^a \\ (u^a)^T & (v^a)^T & \end{bmatrix} \begin{bmatrix} y + \beta t s^y \\ z + \beta t s^z \\ 0 + \beta t \end{bmatrix} - \begin{bmatrix} f_o \\ d \\ d^a \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ h - h \end{bmatrix} = 0$$

$$\Longrightarrow$$

$$\left\{ \begin{bmatrix} K_o & U & u^a \\ U^T & V & v^a \\ (u^a)^T & (v^a)^T & \end{bmatrix} \begin{bmatrix} y \\ z \\ 0 \end{bmatrix} - \begin{bmatrix} f_o \\ d \\ h \end{bmatrix} \right\}$$
$$+ \begin{bmatrix} K_o & U & u^a \\ U^T & V & v^a \\ (u^a)^T & (v^a)^T & \end{bmatrix} \begin{bmatrix} s^y \\ s^z \\ 1 \end{bmatrix} \beta t - \begin{bmatrix} 0 \\ 0 \\ d^a - h \end{bmatrix} = 0 \tag{41}$$

where $h = (u^a)^T y + (v^a)^T z$. In (41), the bracketed quantity is zero because y and z are the solution to (13). The remaining terms can be used to compute s^y and s^z directly using

$$\begin{bmatrix} K_o & U \\ U^T & V \end{bmatrix} \begin{bmatrix} s^y \\ s^z \end{bmatrix} = \begin{bmatrix} -u^a \\ -v^a \end{bmatrix} \tag{42}$$

and the last equation can be used to compute the value of the primal step length $t^P = \gamma^+/\beta = t$ for the full primal step as

$$t^P = \gamma^+/\beta = t = \frac{d^a - (u^a)^T y - (v^a)^T z}{\beta((u^a)^T s^y + (v^a)^T s^z)}. \tag{43}$$

Given the Schur complement S for the current working set, (42) can be solved as

$$s^z = S^{-1}\left(-v^a + U^T K_o^{-1} u^a\right), \tag{44}$$

$$s^y = K_o^{-1}(-u^a - U s^z). \tag{45}$$

Here we see that computing s^y and s^z using (44) and (45) works even if the updated KKT system (40) becomes singular.

To summarize, the two approaches for computing s^y and s^z are given in Algorithms 5.1 and 5.2 below.

Algorithm 5.1. *Compute s^y and s^z by attempting to update the complement $S \to \bar{S}$ first.*

1. *Attempt to update the Schur complement using (25)–(29) which requires a solve with K_o for $r = K_o^{-1} u^p$ if $u^p \neq 0$ except for $\bar{q}^F = q^F - 1$ which involves shrinking S as shown in (32).*
   (a) *If $\bar{S}$ is singular (i.e. because constraint $j^{(+)}$ is linearly dependent with the current working set) then stop this approach and compute s^z and s^y using Algorithm 5.2.*
   (b) *Else $\bar{S}$ is nonsingular and continue to Step 2.*
2. *Solve for $\bar{z}^+$ using (36) which requires a solve with $\bar{S}$.*
3. *Solve for y^+ using (37) which requires a mandatory solve with K_o.*
4. *Compute s^z using (38) and s^y using (39).*

Algorithm 5.2. *Compute s^y and s^z without updating the Schur complement S.*

1. *Compute s^z from (44) which requires a solve with K_o if $u^a \neq 0$ and a solve with S.*
2. *Compute s^y from (45) which requires a mandatory solve with K_o.*

Table 1 shows the counts for the three dominant types of significant computations in an iteration of the dual algorithm for the three types of steps taken in Step $\{4\}$ of Algorithm 3.1. If Algorithm 5.1 is used and if a constraint is dropped in a Partial Primal-Dual step then two separate updates of the factorization of the Schur complement will be performed. If S is indefinite, then this may result in two separate $O(q^3)$ factorizations.

When choosing between Algorithms 5.1 and 5.2 we consider the three types of constraint updates:

- For $\bar{q}^C = q^C + 1$ and $\bar{q}^F = q^F - 1$, $u^a = 0$ and $s^z = -S^{-1} v^a$ in (44), so we can skip a solve with K_o and a matrix-vector multiplication with U. Therefore, Algorithm 5.2 will always be cheaper than using Algorithm 5.1. In addition, if the added constraint is linearly independent, then the update of the Schur complement in step $\{4\}$ of Algorithm 3.1 will not require any further solves with K_o and therefore the total cost of the dual iteration will be essentially the same for Algorithm 5.1 and Algorithm 5.2 (see Table 1). Hence, Algorithm 5.2 is always the better alternative for these cases since it also avoids

Table 1 Counts for the dominant types of computational operations (i.e. number of factorization updates of S, number of solves with K_o, and number of solves with S) for the three types of additions to the working set (i.e. $\bar{q}^F = q^F - 1$, $\bar{q}^C = q^C + 1$, and $\bar{q}^{(+)} = q^{(+)} + 1$) for three different types of iterations of the dual algorithm (i.e. (a) Full Primal-Dual Step, (b) Partial Primal-Dual Step, and (c) Dual Step)

	# Fact. updates of S		# Solves with K_o		# Solves with S	
Type of change	Algo 5.1	Algo 5.2	Algo 5.1	Algo 5.2	Algo 5.1	Algo 5.2
(a) Dual QP iteration ending in Full Primal-Dual Step (add $j^{(+)}$)						
$\bar{q}^F = q^F - 1$	1	1	1	1	1	1
$\bar{q}^C = q^C + 1$	1	1	1	1	1	1
$\bar{q}^{(+)} = q^{(+)} + 1$	1	1	2	3	1	1
(b) Dual QP iteration ending in Partial Primal-Dual Step (drop $j^{(-)}$, add $j^{(+)}$)						
$\bar{q}^F = q^F - 1$	2	1	1	1	1	1
$\bar{q}^C = q^C + 1$	2	1	1	1	1	1
$\bar{q}^{(+)} = q^{(+)} + 1$	2	1	2	3	1	1
(c) Dual QP iteration ending in Dual Step (drop $j^{(-)}$, add $j^{(+)}$)						
$\bar{q}^F = q^F - 1$	2*	1	1	1	1	1
$\bar{q}^C = q^C + 1$	2*	1	1	1	1	1
$\bar{q}^{(+)} = q^{(+)} + 1$	2*	1	4*	3	1	1

*Note, the first Schur complement update (which includes a solve with K_o for $\bar{q}^{(+)} = q^{(+)} + 1$) attempted in Algo 5.1 for the Dual Step iteration fails since $j^{(+)}$ is linearly dependent with the current working set.

a possible initial failure in the update of the factors of S that can occur when using Algorithm 5.1.

- For $\bar{q}^{(+)} = q^{(+)} + 1$ we see from Table 1 that Algorithm 5.1 requires fewer solves if the added constraint $j^{(+)}$ is linearly independent. Otherwise, if the added constraint is linearly dependent then Algorithm 5.2 will always be cheaper than Algorithm 5.1, since the work required to attempt the Schur complement update ($S \rightarrow \bar{S}$ in Algorithm 5.1) will be wasted and Algorithm 5.2 will be called anyway. This occurs, for example, when all of the degrees of freedom are used up (i.e. $n^X - q^F + q^C + q^{(+)} = n$) and the constraint $j^{(+)}$ must be linearly dependent. Here, QPSchur always chooses Algorithm 5.2. On the other hand, if we assume there are not too many linearly dependent constraints, then encountering a singular KKT system should be rare, and the direct update of the Schur complement (Step $\{2\}$ in Algorithm 3.1) will succeed most of the time. Therefore, by default, QPSchur uses Algorithm 5.1 to compute s^z and s^y for the case $\bar{q}^{(+)} = q^{(+)} + 1$ when the degrees of freedom are not used up.

5.2. Initialization of the Schur complement and the warm start algorithm

In many cases, a good guess of the optimal working set is known before invoking the QP solver (e.g. in an rSQP method when near the optimal solution) and by more effectively exploiting this initial guess the overall cost of solving the QP can be greatly reduced. Here we start with a set of fixed and free variables that is used to shrink K_o and we also initialize the Schur complement in (14). The difficulty with the warm start for this QP

algorithm is that an initial guess of the optimal working set may not be dual feasible. Consequently, a simplified (i.e. with respect to the actual implementation) warm start algorithm that adjusts the initial guess of the working set is given in Algorithm 5.3; interfaces and procedures for initializing the Schur complement and dropping constraints are discussed in Bartlett (2001).

Algorithm 5.3. *Dual warm start algorithm for adjusting the initial guess of the working set.*

1. *Initialize* $S = V - U^T K_o^{-1} U$, *factor* S *and remove any constraints that cause* S *to be singular.*
2. *Drop constraints from* $x_L^R \le x^R \le x_U^R$ *and* $c_L \le A_c^T x \le c_U$ *until multipliers in* z *are dual feasible. while (* $q > 0$ *)*
   $z = S^{-1}(d - U^T y_o)$.
   Select multiplier v_j *from* z *with maximum dual infeasibility.*
   if ($|v_j| \le$ *dual_infeas_tol) then exit loop, the multipliers in* z *are dual feasible!*
   Drop constraint j *by shrinking* U, V, d *and* S.
   endwhile
   $y = K_o^{-1}(f_o - Uz)$.
3. *Drop constraints from* $x^X = b^X$ *until multipliers in* μ^D *are dual feasible.*
   while ($q^F < n^X$ *)*
   $(z, y) \rightarrow x^R$.
   $\mu^D = -(Q^D)^T (g^X + (G^{RX})^T x^R + G^{XX} b^X)$.
   Select multiplier μ_j *from* μ^D *with maximum dual infeasibility.*
   if ($|\mu_j| \le$ *dual_infeas_tol) then exit loop, current* μ^D *is dual feasible!*
   Drop constraint j *by augmenting* U, V, d *and* S.
   $z = S^{-1}(d - U^T y_o)$.
   $y = K_o^{-1}(f_o - Uz)$.
   endwhile

We chose the above warm-start algorithm since it makes sense to drop the constraints in S first since updating the multipliers for these constraints only requires a solve with S and not K_o to compute z. On the other hand, updating the multipliers μ^D (for the variables that were initially fixed and left out of K_o in (10)) requires solves with S and K_o and is much more expensive. Therefore, the algorithm only frees as many of these variables as needed (which requires that the Schur complement be augmented each time).

The key to a highly efficient implementation of the warm start algorithm above is in the formation and factorizaton of the initial Schur complement $S = V - U^T K_o^{-1} U$ in Algorithm 5.3 Step $\{5.3\}$. The details on computing $S_o = U^T K_o^{-1} U$ are up to the matrix object that implements K_o. When a symmetric direct factorization is used for $K_o = L_o D_o L_o^T$, then the above warm start algorithm will only require one backsolve with L_o per general active constraint instead of four backsolves for a cold start. In addition, level-3 BLAS can be used for the block backsolve $L_o^{-1} U$ instead of just level-2 BLAS when S is updated one column at a time. Therefore, one would expect a warm start to be more than four times faster than a cold start when there are many active inequalities and the working set does not change much from the initial guess. Results showing this behavior are given in Section 6.1. Additional details of this the QP algorithm and refinements can be found in Bartlett (2001).

5.3. Dealing with degeneracy, ill-conditioning and roundoff errors

Here, we briefly mention some numerical implementation details described more fully in Bartlett (2001). Roundoff errors in floating point computations, which are magnified by ill-conditioning in the underlying QP, can cause problems in many different areas of the algorithm and therefore require special attention. Even if the KKT system for the current working set is well conditioned, ill-conditioning of K_o will compromise the accuracy of the computed solutions. This is because K_o is used to update S and roundoff errors are carried over into the formation of the Schur complement even before S is factored.

Our primary strategy for combating this inaccuracy is to use fixed-precision iterative refinement given K_o and the factorized S. While this is not expected to improve the accuracy of the computed solution when Gaussian elimination with full row or column pivoting is used (see Section 3.5.3 in Golub, 1996), it may be effective for the Schur complement method. The reason for this is that roundoff in the Schur complement method can be much more significant than in Gaussian elimination with unrestricted complete or partial pivoting, as pivoting in a Schur complement method is only allowed separately within K_o and S and not between these matrices. For most reasonably conditioned nondegenerate QPs, iterative refinement is not needed to identify the optimal working set. But, it can be used at the solution to help improve the solution accuracy without impacting runtime too much. Moreover, iterative refinement can be especially useful for:

- correcting a constraint in the current working set that may be violated by more than the allowed tolerance,
- correcting very small multipliers with an incorrect sign. Here, if the dual feasibility violation is larger than a tolerance, then iterative refinement is applied.
- correcting for the wrong sign of γ^+ due to roundoff errors in the denominator of (43). This can occur when a linearly dependent constraint $j^{(+)}$ is being added to the working set and $|\gamma^+| \approx \infty$.

Also, if there is degeneracy in the working set, or after the algorithm is terminated, it is desirable to recompute the solution using equations like (15)–(16) and (23) to eliminate accumulated roundoff errors. Applying iterative refinement works very well in practice and is cheap enough to be the default mode for QPSchur. Finally, numerous additional floating-point issues that are critical to the success of the implementation have also been incorporated within QPSchur and are described in Bartlett (2001).

6. Numerical results

This section illustrates the performance of QPSchur on a variety of test problems drawn from Model Predictive Control (MPC) and from reduced-space Successsive Quadratic Programming (rSQP). This section does not provide a detailed benchmarking with the purpose of selecting the best QP solver. Rather our intention is to demonstrate the efficiency and flexibility of the dual, Schur Complement algorithm coupled with an object oriented implementation. To address this point we consider QP classes derived from banded as well as rSQP structures. For the numerical study we consider two other QP solvers, QPOPT (Gill et al., 1995) and QPKWIK (Schmid and Biegler, 1994) as well as a number of options and specializations for QPSchur. Other structures, such as sparse Hessians and constraint gradients

exploited in LOQO (Vanderbei, 1994) and SOCS (Betts and Frank, 1994), could also be considered but direct sparse matrix solvers have not yet been incorporated within QPSchur and remain a topic for future study.

Our numerical results are divided into three categories:

- Solution of large QPs drawn from a specialized banded Model Predictive Control application. In this comparison we demonstrate the flexibility of QPSchur in adapting to the specialized structure of a large QP problem.
- Solution of QP subproblems within an rSQP algorithm, applied to a scaleable nonlinear program. In this comparison we compare different approaches that incorporate reduced Hessian information. This is particularly easy with the object-oriented implementation of QPSchur.
- Solution of QP subproblems within an rSQP algorithm (Schmid and Biegler, 1994) applied to a set of test problems drawn from the CUTE collection. Here we also examine the iterative refinement option in QPSchur to enhance its reliability.

6.1. Model Predictive Control

We first consider a structured quadratic program described in Bartlett et al. (2002); this is a large model predictive control problem (MPC) for cross-directional control of a paper machine. Here a quadratic objective function is used to drive the output variables to their setpoints and also regularize the input variable profiles. For the purpose of this study we consider an input horizon of one with a range of n_u =150 to 1200 input variables. Because of the large size, the speed and robustness of the QP solver is critical in this application. As detailed in Bartlett et al. (2002), the QP is expressed only in terms of the input variables; it contains only one equality constraint, n_u general process-specific inequality constrains and bounds on all n_u input variables. Because of the special structure of the paper machine model, the Hessian G is a symmetric positive definite banded matrix (bandwidth = 15), which can be formed and factored offline using standard banded LAPACK subroutines. The A_c matrix in (2) is also banded, with a bandwidth of two. The banded structure of A_c is fully exploited by the Schur complement method implemented in QPSchur through banded BLAS subroutines.

Table 2 MPC QP sizes and **solution statistics** for cold starts where: 'n_u' = number of manipulated variables, 'obj' = objective function value, '#bs' = number of active variable bounds at the solution, '#in' = number of active general inequality constraints at the solution. Note that different numbers of active constraints for case2 where reported for each QP solver which was due to ill conditioning and degeneracy at the solution

		QPOPT			QPKWIK			QPSchur		
	n_u	obj	#bs	#in	obj	#bs	#in	obj	#bs	#in
case1a	150	−959.394	0	1	−959.394	0	1	−959.394	0	1
case1b	300	−2178.39	0	1	−2178.39	0	1	−2178.39	0	1
case1c	600	−3603.92	0	1	−3603.92	0	1	−3603.92	0	1
case1d	1200	−7211.06	0	1	−7211.06	0	1	−7211.06	0	1
case2	600	−34082.7	438	141	−34082.7	431	148	−34082.7	453	126
case3	600	−1617.47	0	0	−1617.47	0	0	−1617.47	0	0
case4	600	−1616.95	0	3	−1616.95	0	3	−1616.95	0	3

Table 3 MPC QP solution iteration counts and CPU times for **cold starts** on a Linux 800 MHz Intel PIII with g++/g77 where : 'iter' = number of QP iterations, 'D/D' = CPU time (sec) using a dense Hessian and dense Jacobian, 'B/D' = CPU time (sec) using a banded Hessian and dense Jacobian, and 'B/B' = CPU time (sec) using a banded Hessian and banded Jacobian

	QPOPT			QPKWIK		QPSchur			
	iter	D/D	B/D	iter	D/D	iter	D/D	B/D	B/B
case1a	2	0.23	0.15	2	0.13	2	0.12	0.11	0.11
case1b	2	1.46	0.94	2	0.41	2	0.29	0.24	0.22
case1c	2	10.57	6.67	2	1.92	2	0.94	0.54	0.44
case1d	2	76.67	43.43	2	12.49	2	5.16	1.21	0.92
case2	951	41.44	31.03	1104	44.39	900	32.28	14.87	8.91
case3	1	10.43	6.56	1	1.85	1	0.86	0.48	0.44
case4	22	11.34	7.43	4	2.04	4	1.04	0.58	0.45

Table 4 MPC QP solution iteration counts and CPU times for **warm starts** on a Linux 800 MHz Intel PIII with g++/g77 where : 'iter' = number of QP iterations, 'D/D' = CPU time (sec) using a dense Hessian and dense Jacobian, 'B/D' = CPU time (sec) using a banded Hessian and dense Jacobian, and 'B/B' = CPU time (sec) using a banded Hessian and banded Jacobian

	QPOPT			QPKWIK		QPSchur			
	iter	D/D	B/D	iter	D/D	iter	D/D	B/D	B/B
case1a	0	0.24	0.15	2	0.14	0	0.13	0.12	0.12
case1b	0	1.46	0.93	2	0.44	0	0.32	0.25	0.23
case1c	0	10.49	6.94	2	2.15	0	0.94	0.68	0.47
case1d	0	76.71	45.05	2	13.07	0	5.26	1.37	0.98
case2	45	1.78	1.28	1036	31.91	0	4.89	7.58	1.15
case3	1	6.36	2.2	0	2.04	0	0.93	0.52	0.48
case4	1	10.7	6.68	4	2.14	0	0.94	0.06	0.46

Seven scenarios with different QP vectors were generated in Bartlett et al. (2002) and these are considered in Tables 2 to 4 when comparing the three active-set QP solvers: QPSchur, QPKWIK and QPOPT. QPOPT can only exploit the banded Hessian G and not the banded Jacobian A_c. QPKWIK (the Goldfarb and Idnani implementation) cannot exploit any structure in G or A_c. On the other hand, QPSchur fully exploits the banded structure of G and A_c (i.e. by calling specialized BLAS and LAPACK routines for banded matrices). Table 2 provides the problem and solution statistics for the seven scenarios and indicates that all of the methods solved the problems successfully. Table 3 gives the iteration counts and solution times for the three QP solvers using cold starts. Here we see that the fully banded version of QPSchur(B/B) has the best results. In fact, for cases with few active constraints (i.e. case1a - case1d, case3, case4) this implementation fully exploits the structure and properties of the QP. Also note that the cost for QPSchur(B/B) increases only linearly with n_u from case1a to case1d, as a result of linear algebra done with banded matrices. Table 4 gives the iteration counts for warm starts which were produced, given the solutions from the cold start runs. With few active inequality constraints in case1, case3 and case4, there is little difference

between Tables 3 and 4 between the runtimes for QPOPT, QPKWIK and QPSchur. However, for case2 (i.e. where there are many active inequalities) the reductions in runtime for QPOPT and QPSchur using warm starts are significant.

These results show the significance of exploiting the problem structure and utilizing warm start information. They show that by using QPSchur(B/B) (i.e. with specialized banded matrix objects) all of the test QPs were solved much faster than with the other solvers.

6.2. Scaleable problem for rSQP

In Successive Quadratic Programming (SQP) methods, the solution to a general Nonlinear Program (NLP) is sought by solving a sequence of QPs. A particular class of SQP methods (referred to as rSQP methods) seeks to solve the QP subproblems using projections into the linearized equality constraints. For a nonlinear program with n variables and m equality constraints, these projections allow the Hessian of the Lagrangian to be represented in a reduced space of dimension $n - m$. Moreover, in many applications, this matrix is approximated using quasi-Newton methods (e.g. BFGS (Nocedal and Wright, 1999)). As described in Schmid and Biegler (1994) the reduced space QP subproblem takes the form

$$\min \quad g^T p^z + \tfrac{1}{2}(p^z)^T B p^z + (\eta^2/2 + \eta)M \tag{46}$$

$$\text{s.t.} \quad b_L \leq \begin{bmatrix} E \\ I \end{bmatrix} p^z + b\eta \leq b_U \tag{47}$$

where $B \in \mathbb{R}^{(n-m)\times(n-m)}$ is s.p.d., $g \in \mathbb{R}^{n-m}$, $M \in \mathbb{R}$, $E \in \mathbb{R}^{m\times(n-m)}$, and $b_L, b_U \in \mathbb{R}^n$. Both B and E are usually dense matrices but may also have very specialized structure and/or properties Also, the variable η is added to allow a relaxation of the inequality constraints, as $p^z = 0$ and $\eta = 1$ is a feasible point if $b^L \leq b \leq b^U$. The relaxation variable η is penalized in the objective function using a large constant M. Since we expect that the QP will be feasible, the relaxation variable $\eta = 0$ is initially fixed and left out of the initial KKT system, K_o, in QPSchur.

Here the quasi-Newton approximation for B is provided through a s.p.d. BFGS approximation (Nocedal and Wright, 1999). When $n - m$ is small, dense matrix approximations are inexpensive and lead to good performance. However, dense approximations can be prohibitively expensive as $n - m$ becomes large. We therefore consider the following options with our object-oriented implementation of QPSchur used within an rSQP algorithm (Bartlett, 2001):

- **BFGS:** Using the BFGS formula, we directly update and store the Cholesky factors of $K_o = B = LL^T$. Updating the Cholesky factor and solving the linear system (i.e. solves with K_o) both require $O((n-m)^2)$ flops.
- **LBFGS:** We apply a limited memory approximation for $K_o = B$ where the ℓ most recent BFGS updates are used (Byrd et al., 1994). Normally $\ell \ll (n-m)$. LBFGS updates and linear systems solves require only $O((n-m)\ell)$ flops. However, using the LBFGS matrix only gives linear convergence while the dense BFGS gives superlinear convergence in the rSQP algorithm.
- **PBFGS, PLBFGS:** As discussed in Bartlett (2001), one can also approximate only a subset of B using the BFGS method by noting which variables remain at their bounds and contribute no new update information. Using the permutation matrix $Q = [Q^R \;\; Q^X]$ we select the initially free and fixed reduced QP variables and update only the 'free'

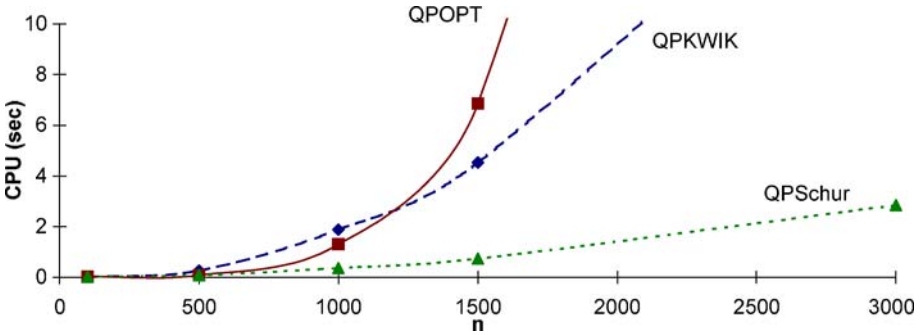

Fig. 1 CPU times for QP solvers (warm starts, dense BFGS) and overall rSQP iteration time for NLP (48)–(50) with nact = 50 active variable bounds ($n - m = n/2$)

reduced Hessian $K_o = B^{RR} = (Q^R)^T B(Q^R)$ using either the BFGS or LBFGS options. For each solve with K_o we require only $O((n - m - n^X)^2)$ flops with PBFGS and only $O((n - m - n^X)\ell)$ flops with PLBFGS.

To compare the linear algebra costs for the reduced Hessian B options and the choice of the initial KKT system K_o in QPSchur, we consider the simple scaleable test problem

$$\min \quad \tfrac{1}{2} x^T x \tag{48}$$

$$\text{s.t.} \quad c_j = x_j(x_{(j+n/2)} - 1) - 10x_{(j+n/2)} = 0, \quad \text{for } j = 1 \ldots n/2 \tag{49}$$

$$x_i \geq 0.01, \quad \text{for } i = 1 \ldots nact. \tag{50}$$

The dimension of the reduced space for this NLP is $(n - m) = n/2$ and the number of active inequality constraints at the solution can be scaled from $nact = 0$ up to $nact \leq (n - m)$.

In the first set of numerical tests shown in Figure 1, we compare the efficiency of QPSchur, QPOPT and QPKWIK using a dense BFGS matrix B. For QPSchur, the Cholesky factor $B = LL^T$ was directly updated by the rSQP algorithm. For QPKWIK, the inverse Cholesky factor $B^{-1} = L^{-T}L^{-1}$ is updated and for QPOPT, the upper triangular part of B is updated. Figure 1 shows the CPU times per rSQP iteration using dense BFGS with $nact = 50$, a warm start for the working set and increasing n for the NLP. Here QPOPT has a complexity of $O((n - m - nact)^3)$ which explains the rapid increase in runtime for larger QPs with few active inequality constraints. With a dense BFGS Hessian both QPKWIK and QPSchur are $O((n - m)^2 nact)$ but, due to an efficient implementation of the Schur complement and the warm-start algorithm described in Section 5.2, QPSchur requires only $1/6$ times the flops of QPKWIK.

Figure 2 compares QPOPT and QPSchur using both the dense BFGS approximation and the LBFGS (with $\ell = 2$) options. Here QPOPT also uses a compact LBFGS representation for B (as opposed to B^{-1} used with QPSchur); this is also $O((n - m)p)$. As shown in the figure, the LBFGS B matrix made little difference with QPOPT since $O((n - m - nact)^3)$ internal Cholesky factorizations dominate its runtime. However, the impact of using LBFGS with QPSchur is dramatic; LBFGS drops the runtime complexity per rSQP iteration from $O((n - m)^2 nact)$ to $O((n - m)\,\ell\,(nact))$.

Finally, for large $nact$ even the $O((n - m)\,\ell\,(nact))$ flops performed by QPSchur with LBFGS dominate the cost of the rSQP algorithm and this requires the projected BFGS

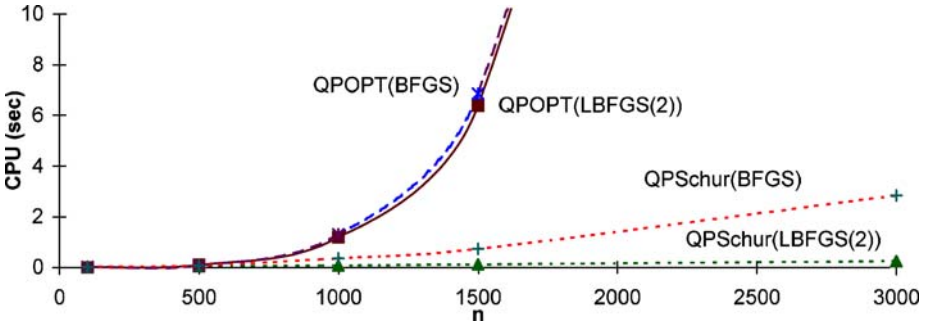

Fig. 2 CPU times for QP solvers (warm starts, LBFGS(2)) and overall rSQP iteration time for NLP in (48)–(50), nact = 50 active variable bounds ($n - m = n/2$)

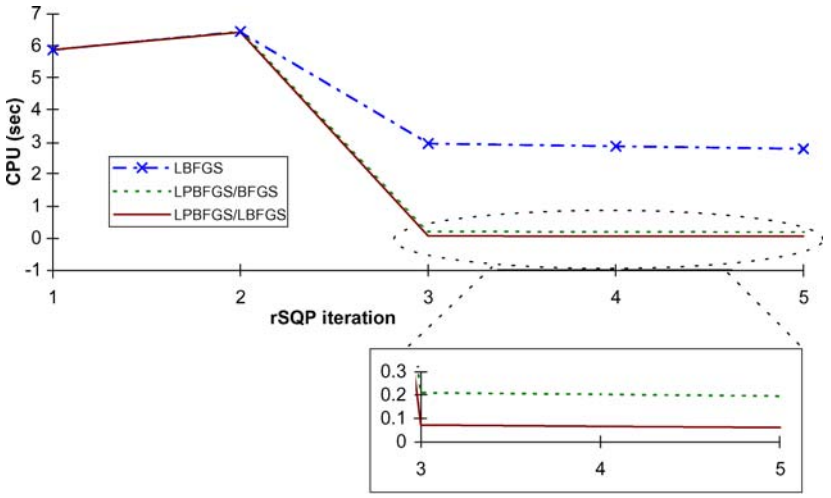

Fig. 3 CPU times for rSQP iterations using QPSchur with LBFGS and LPBFGS for NLP (48)–(50) with $n = 2000$ ($n - m = n/2$) and nact = 500 active variable bounds. Enlargement of plot for rSQP iterations 3, 4 and 5 is also shown

approximation, B^{RR}. Figure 3 shows the overall CPU runtimes per rSQP iteration using LBFGS for the first two iterations and then switching to projected BFGS updating for the last three iterations. The dramatic reduction in runtime for the projected BFGS updating (PBFGS and PLBFGS) over LBFGS is due to the fact that $K_o = B^{RR}$ is used and therefore, the Schur complement remains empty in this scenario. However, $n^R = 500$ is still quite large and the enlargement in Figure 3 also shows a savings in using PLBFGS over PBFGS.

These results demonstrate the flexibility of QPSchur. Simply by replacing the matrix object that represents the initial KKT matrix K_o, it is possible to dramatically improve the efficiency of the QP solver. For specialized classes of QPs that arise in MPC and rSQP, the Schur complement QP solver very effectively exploits the structure of the problem at hand.

Table 5 Results for using QPKWIK, QPOPT and QPSchur to solve the set of test NLPs (max_iter = 100). There were $2^4 = 16$ (four options varied) runs per NLP with each QP solver: 'solved' = number of rSQP runs using the QP solver that converged, 'max_iter' = number of rSQP runs that did not converge before the maximum iteration count of 100, 'except' = number of rSQP runs where an unrecoverable error occurred.

QP Solver	solved	max_iter	except
QPKWIK	574	136	10
QPOPT	589	131	0
QPSchur	602	118	0

6.3. rSQP comparison with CUTE test problems

Finally, we consider QPSchur within an rSQP algorithm on 45 NLPs drawn from the Hock-Schittkowski/CUTE test set and compare it with QPOPT and QPKWIK. These 45 test problems were selected because each has at least one general constraint and at least one active inequality constraint at the solution; HS11–HS14, HS16–HS24, HS29–HS37, HS41, HS42, HS43–HS44, HS59, HS64, HS66, HS70–HS76, HS83, HS86, HS100, HS105, HS107–HS109, HS114, HS118, HS119 (see Bartlett (2001)). Using all possible combinations of four different rSQP options ($2^4 = 16$ combinations), a total of $45(16) = 720$ NLP runs were generated with up to 100 QP subproblems solved per rSQP run (`max_iter` = 100). Therefore, these tests represent the solution of thousands of QP subproblems and demonstrate the reliability of QPSchur. Table 5 compares QPKWIK, QPOPT and QPSchur for these test NLPs. In exact arithmetic all of these solvers should produce the same rSQP iterations and should differ only in runtimes. However, with ill-conditioned NLPs solved using floating point arithmetic, the number of rSQP iterations can vary significantly based on the choice of QP solver and internal tolerances. It should be noted that several devices were in place for the rSQP algorithm to minimize the number of algorithm failures. For example, the rSQP algorithm was reinitialized in many cases when the QP solver failed. Therefore, several of the runs where `max_iter` = 100 was exceeded were actually repeated failures with subsequent algorithm reinitializations. The results in Table 5 indicate that QPSchur is fairly stable, as it solved more NLPs than QPKWIK or QPOPT. It is possible that better results may be obtained for QPKWIK and QPOPT with some tolerance adjustment. Also, by increasing `max_iter` to 1000, additional NLP runs converged, but more failures were reported as well.

Finally, these test NLPs present an opportunity to study the importance of iterative refinement in QPSchur. Table 6 shows results for QPSchur using two different rSQP safeguard options (see Bartlett (2001)). In the context of this paper, what is important about these safeguard options is that when turned on, these safeguards reduce the number of ill conditioned QP subproblems presented to QPSchur. When the basis repartitioning (BC) and Hessian reinitialization (RH) safeguards are enabled in rSQP, disabling iterative refinement within QPSchur has little impact; 602 NLPs are solved for (IR,BC,RH) instead of 601 for (nIR,BC,RH). However, when these safeguards are turned off, the number of NLP failures increases dramatically. The difference between using iterative refinement or not is most significant when both safeguards are disabled (i.e. compare (IR, nBC, nRH) with (nIR, nBC,

Table 6 Results for using QPSchur to solve the set of test NLPs (max_iter = 100). There were $2^4 = 16$ (four options varied) runs per NLP attempted: 'solved' = number of rSQP runs that converged, 'max_iter' = number of rSQP runs that did not converge before the maximum iteration count of 100, 'except' = number of rSQP runs where an unrecoverable error occurred. 'IR' = Iterative refinement with QPSchur was allowed, 'nIR' = Iterative refinement with QPSchur was not allowed, 'BC' = Test for basis ill conditioning and repartitioning enabled, 'nBC' = Test for basis ill conditioning and repartitioning disabled, 'RH' = Reinitialization of Hessian on QP failure, 'nRH' = Algorithm exception on QP failure.

	solved	max_iter	except
IR,BC,RH	602	118	0
nIR,BC,RH	601	117	2
IR,nBC,RH	587	122	11
IR,BC,nRH	591	109	20
IR,nBC,nRH	573	110	37
nIR,nBC,nRH	555	74	91

nRH)). Without these safeguards, QPSchur is asked to solve more poorly conditioned QP subproblems and reports more failures. In particular, comparing the cases (IR,nBC,nRH) and (nIR,nBC,nRH) shows that QPSchur can solve more QPs with iterative refinement, although it is not always successful. When iterative refinement is used, the number of rSQP algorithm exceptions drops from 91 to 37.

7. Conclusions and future work

We develop a dual-feasible active-set strategy for solving quadratic problems. Addition and dropping of constraints is provided through a Schur complement updating strategy. A detailed overview is provided and several challenging applications are presented that demonstrate the performance of this approach. We also mention that this method is readily extended to exploit equality constraints of the form $A_e x = b$. Here we simply define $K_o = \begin{bmatrix} G & A_e \\ A_e & 0 \end{bmatrix}$ and proceed as above. A complete implementation of this approach is described in Bartlett (2001).

Considerable planning and experimentation have gone into developing the QPSchur formulation, algorithm and its object-oriented software implementation. Not only can QPSchur be used to exploit the properties of many types of specialized QPs efficiently, but these QPs can be solved using the same core QPSchur implementation. In contrast, a QP solver developed for a specialized application area must deal with all of the detailed implementation pitfalls from scratch.

However, there is still more to be done with QPSchur that would increase its efficiency and robustness in the following areas: (i) implementation of factorizaton updating strategies for symmetric indefinite Schur complements, (ii) generalized selection strategies to add violated

constraints and (iii) the incorporation of abstract vectors for parallel and other specialized applications and computing environments.

(i) As was previously mentioned, the efficiency of this QP algorithm relies strongly on the implementation of the Schur complement (i.e. updating and downdating). When the Schur complement is positive or negative definite the current Cholesky implementation is very efficient. However, when the Schur complement is very large and indefinite, the current Bunch-Kaufman implementation, which refactors S from scratch for each change, becomes significantly less efficient. QPSchur would greatly benefit from a cheaper but more complex implementation of updating an indefinite symmetric matrix (i.e. along the lines of Sorensen (1977)).

(ii) The current implementation of the constraints class uses only a few simple strategies for selecting violated constraints for the dual QP algorithm. When there are relatively few constraints or when the constraints are sparse then it is comparatively cheap to compute the residual of all the constraints when looking for violated constraints. However, other QP applications may have many more general inequality constraints than variables and these constraints may be fairly dense (e.g. rSQP where $(n - m) \ll m$). In these cases, evaluating every constraint to look for a violated constraint would dominate the runtime to solve the QP. A more sophisticated constraint selection strategy would keep a list of likely violated constraints that would be checked at each iteration. Such a strategy would try to reach a balance between reducing the number of constraint evaluations per QP iteration while trying not to pick constraints with small violations that are likely to be freed later on.

(iii) Up to this point, the implementation of QPSchur has focused on trying to abstract matrices such as G, A and K_o away from the core algorithmic code so that the structure and special properties of these matrices and the QP application area can be exploited. In particular, we intend to provide capabilities for direct sparse and iterative linear solvers. As long as the executables operate in a strictly shared-memory environment then every piece of data is stored in local memory and this implementation is satisfactory. However, with more sophisticated computing environments such as distributed-memory multi-processor machines that use using parallel algorithms, QPSchur must be modified to reach its full potential in these special environments. Here serial representations of vectors in QPSchur must be replaced with abstract vector interfaces, as this would allow for efficient implementations when used in these parallel and other specialized environments, as well as in simple shared-memory situations (Bartlett et al., 2003). Work in this area is currently underway.

References

Bartlett, RA (2001) Object oriented approaches to large-scale nonlinear programming for process systems engineering. PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA

Bartlett RA, Biegler LT, Backstrom J, Gopal V (2002) Quadratic programming algorithms for large-scale model predictive control. J. Process Control 82:775

Bartlett RA, Biegler LT (2003) RSQP++: An object-oriented framework for successive quadratic programming. In Large-scale PDE-Constrained Optimization, Lecture Notes in Computational Science and Engineering 30, Berlin, Springer Verlag, pp 316

Betts JT, Frank PD (1994) A sparse nonlinear optimization algorithm. J. Opt. Theory Appl. 82(3):519–541

Byrd RH, Nocedal J, Schnabel RB (1994) Representations of quasi-Newton matrices and their use in limited methods. Math. Prog. 63:129–156

Carlson D (1986) What are Schur complements, anyway? Lin. Alg. Appl. 74:257–275

Fletcher R (1981) Practical methods of optimization. John Wiley & Sons
Gill P, Murray W, Saunders M (1995) User's Guide for QPOPT 1.0: A Fortran Package for Quadratic Programming. Systems Optimization Laboratory, Department of Operations Research, Stanford University
Gill PE, Murray W, Saunders MA, Wright MH (1990) A Schur complement method for sparse quadratic programming. In Reliable Numerical Computation, Oxford University Press, pp 113–138
Goldfarb D, Idnani A (1983) A numerically stable dual method for solving strictly convex quadratic programs. Mathematical Programming 27:1–33
Golub GH, Van Loan CF (1996) Matrix computations, third edition. Johns Hopkins University Press
Nash S, Sofer A (1996) Linear and nonlinear programming. McGraw Hill
Nocedal J, Wright S (1999) Numerical optimization. Springer, New York
Powell M (1983) ZQPCVX: A Fortran subroutine for convex quadratic programming. Technical report, Department of Applied Mathematics and Theoretical Physics, Cambridge University
Schmid C, Biegler LT (1994) Quadratic programming methods for reduced hessian SQP. Comp. Chem. Eng. 18:817
Sorensen DC (1977) Updating the Symmetric Indefinite Factorization with Applications in a Modified Newton Method. Technical Report ANL-77-49, Argonne National Laboratory
Vanderbei RJ (1994) An Interior Point code for Quadratic Programming. Technical Report SOR 94-15, Princeton University